Microcontroller Pocket Book (Arduino)

Kafemikrochip

2025

Copyright Page

Book title	Microcontroller Pocket Book
Author	Eko Travada Suprapto Putro, ST, MT
Publication Year	2025
Edition	First Edition
Publisher	CafeMikrochip
Website	Kafemikrochip.com

License to Use

List of contents

list of Figures

Arduino Basics Learning Guide

Introduction

Welcome to the world of Arduino! This guide is designed to help you, the beginner, understand the basic concepts and start your journey in creating interactive electronics projects using the Arduino platform. Arduino is an amazing open-source platform consisting of a microcontroller board (hardware) and easy-to-use software (IDE), allowing anyone from artists to engineers to bring their creative ideas to life.

This guide will take you step by step, from an introduction to the Arduino board itself, basic electronic components that are often used, the basics of programming with simplified C/C++, to practical implementations of controlling LEDs, reading buttons, using common sensors (temperature, light, proximity), and displaying information to an LCD screen. At the end of the guide, there are competency test questions to measure your understanding.

Let's start this exciting adventure and turn your ideas into reality with Arduino!

Chapter 1: Getting to Know the Arduino Board

Welcome to the world of Arduino! If you're just starting out with electronics and programming, Arduino is a fantastic starting point. The platform is designed to make it easy for anyone from artists, designers, hobbyists, and engineers to create interactive projects ranging from simple robots to complex home automation systems. This chapter will introduce you to the basic concepts of Arduino, the main components found on its boards, the different types of boards available, and the software used to program them.

What is Arduino?

Arduino is an open-source electronics prototyping platform. This means that the Arduino hardware and software designs are freely accessible, modifiable, and redistributable by anyone. The core of the platform is a combination of a physical circuit board (often referred to as an "Arduino board") that contains a microcontroller, and software in the form of an Integrated Development Environment (IDE) that runs on your computer. This IDE is what is used to write and upload program code to the Arduino board.

Flexibility and ease of use are the main keys to Arduino's popularity. Its goal is to simplify the process of building electronic prototypes, so that even individuals without a deep technical background can start experimenting and bringing their creative ideas to life. Supported by a large and active global community, Arduino users can easily find a wide variety of sample projects, tutorials, and ready-to-use code libraries to speed up development.

Main Components of an Arduino Board (Example: Arduino UNO)

While there are many different types of Arduino boards, most have similar core components. Let's take a look at the main components of the Arduino UNO, one of the most popular and recommended boards for beginners:

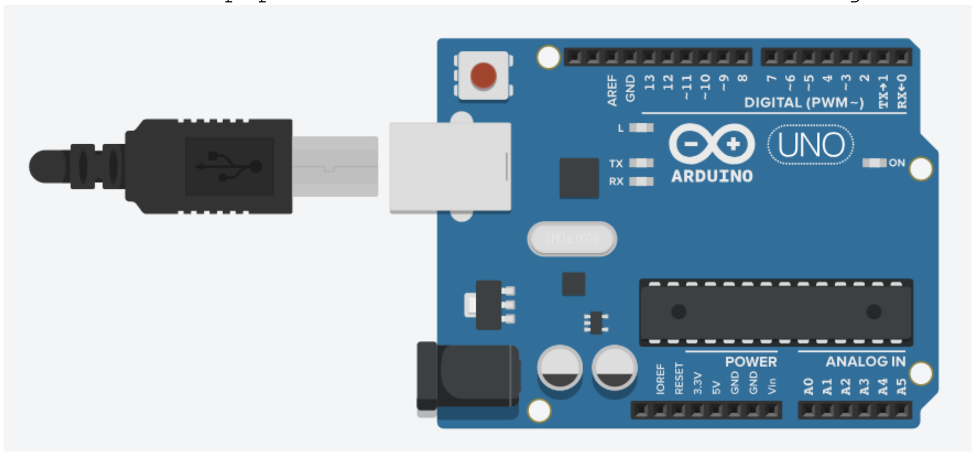


Figure 1 Arduino Uno (R3)

(ThinkerCad, 2025)

1. Microcontroller: This is the "brain" of the Arduino board. On the Arduino UNO R3, the microcontroller used is the ATmega328P. This small IC (Integrated Circuit) is what runs the program code you upload, reads inputs, and controls outputs.
2. Input/Output (I/O) Pins: These are the pins along the edge of the board that allow the Arduino to interact with the outside world.
    a. Digital Pins (0-13): Can function as inputs (reading HIGH/LOW digital signals, such as from a button) or outputs (sending HIGH/LOW digital signals, such as to light an LED). Some digital pins (marked with a '~') also support *Pulse Width Modulation* (PWM), which allows for pseudo-analog output (for example to control LED brightness or motor speed).
    b. Analog Input Pins (A0-A5): Used specifically to read analog signals (varying voltages, not just HIGH/LOW), such as from a temperature or light sensor (LDR). These pins have an internal Analog-to-Digital converter (ADC).
3. Power Pins: Provides various voltage levels to power external components.
    c. VIN: Pin for providing external power to the Arduino (other than via USB or DC jack).
    d. 5V: Regulated 5 Volt output voltage, commonly used to power sensors and other components.
    e. 3.3V: Regulated 3.3 Volt output voltage, required by some sensors and modules.
    f. GND (Ground): 0 Volt reference pin. All components in a circuit must be connected to the same ground.
    g. IOREF: Provides the I/O board operating voltage reference (e.g. 5V for UNO).
4. USB Connector: A type B port (like on a printer) used to connect the Arduino to a computer. This connection is used to upload programs and provide power to the board, as well as for serial communication between the Arduino and the computer.
5. DC Power Jack: Allows the Arduino to be powered by an external power source (AC-to-DC adapter or battery) with a voltage range of typically 7-12 Volts.
6. Voltage Regulator: A component that takes voltage from the DC jack or VIN pin and steps it down to the stable 5V and 3.3V voltages required by the microcontroller and output pins.
7. Reset Button: A small button to restart the program execution from the beginning (same as unplugging and replugging the USB cable).
8. Built-in LED: A small LED connected to one of the digital pins (usually pin 13 on the UNO). Very useful for basic program testing without the need to connect external components.
9. Indicator LEDs (TX, RX, Power): Small LEDs that indicate power status (ON) and serial communication activity (TX - Transmit, RX - Receive).
10. Crystal Oscillator: Generates a precise clock signal (usually 16 MHz on the UNO) to control the speed of instruction execution by the microcontroller.

Types of Arduino Boards

Arduinos come in all shapes and sizes, each with different features and specifications to meet the needs of a variety of projects. Here are some popular Arduino families and boards:

1. Classic Family (Example: Arduino UNO R3): Shown in Figure 1.1. This is the "standard" and most iconic board. The Arduino UNO R3 is highly recommended for beginners because of its balance between pin count, ease of use, and wealth of learning resources available. Other boards in this family include the Leonardo (with built-in USB HID capabilities) and the Micro (a smaller version of the Leonardo).
2. Nano Family: Very small footprint boards, suitable for projects that require compact dimensions. Examples include the Nano Every (affordable basic version), Nano 33 IoT (with Wi-Fi connectivity), Nano 33 BLE Sense (with Bluetooth® Low Energy and integrated sensors such as temperature, humidity, pressure, gestures, microphone), and Nano RP2040 Connect (using the Raspberry Pi RP2040 chip with Wi-Fi/Bluetooth®).

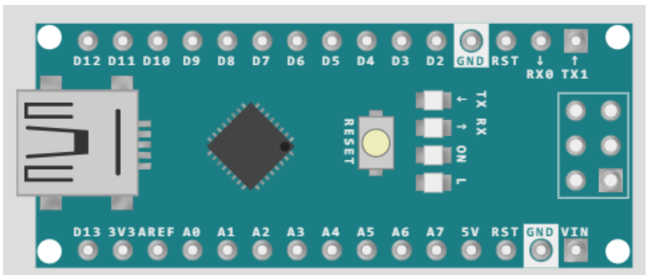

D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
RESET
TX RX ON L
D13 3V3 AREF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN

Figure 2 Arduino Nano *(Wokwi, 2019)*

3. Mega Family (Example: Arduino Mega 2560 Rev3): Designed for more complex projects that require significantly more I/O pins (over 50 digital pins) and more memory than the UNO. The Arduino Due also falls into this category, using a more powerful 32-bit ARM-based microcontroller (but operating at 3.3V).

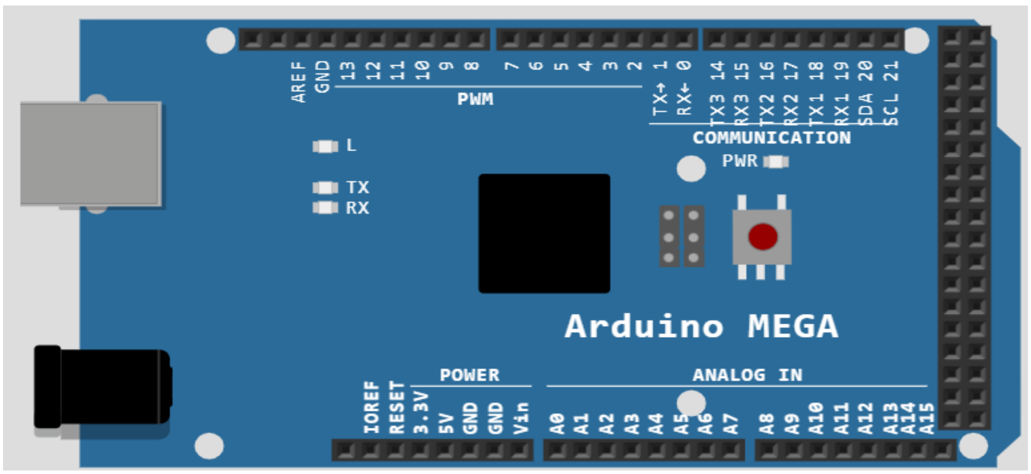

AREF GND 13 12 11 10 9 8 7 6 5 4 3 2 TX→1 RX←0 TX3 14 RX3 15 TX2 16 RX2 17 TX1 18 RX1 19 SDA 20 SCL 21
PWM
COMMUNICATION
L
TX
RX
PWR
Arduino MEGA
POWER
ANALOG IN
IOREF RESET 3.3V 5V GND GND Vin
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15

Figure 3 Arduino Mega *(Wokwi, 2019)*

4. MKR Family: A series of modular boards designed specifically for IoT (Internet of Things) and connectivity projects. MKR boards typically
5. equipped with integrated radio modules (such as Wi-Fi, Bluetooth®, LoRa®, Sigfox®, NB-IoT) and crypto chips for secure communication. A variety of MKR *shields* and *carriers* are also available to easily add functionality.

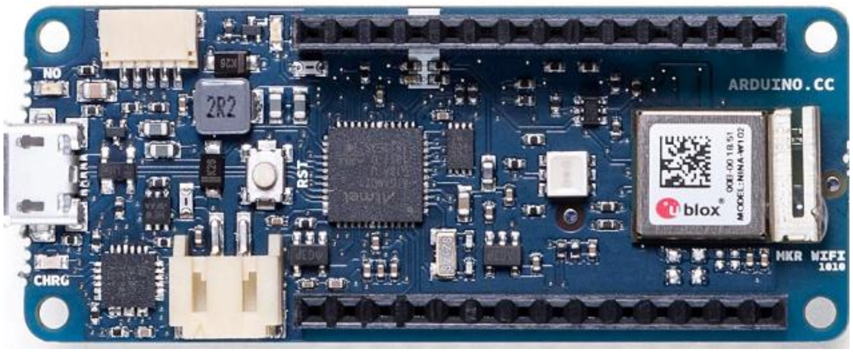


Figure 4 MKR Board *(Arduino, 2025)*

6. ESP-Based Boards: While not officially under the "Arduino" brand, boards such as the ESP8266 (NodeMCU) and ESP32 are very popular among the Arduino community due to their low cost, integrated Wi-Fi and Bluetooth®, and ability to be programmed using the Arduino IDE after installing additional board support.

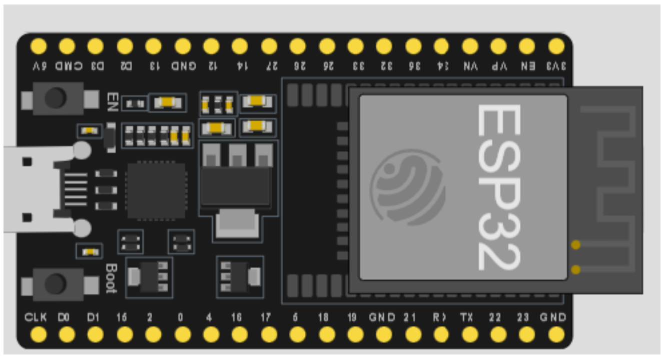


Figure 5 ESP 32 Board *(Wokwi, 2019)*

In addition to official Arduino boards, there are also *clone* boards (unofficial copies that are often cheaper) and *compatible* boards (made by other manufacturers but designed to be compatible with the Arduino ecosystem). While clone boards can be a more economical alternative, their quality and compatibility can sometimes vary.

Choosing the Right Arduino Board for Beginners

For beginners, the Arduino UNO R3 (or a good quality clone) is generally the best choice. Here's why:

- Popular: Most widely used, so tutorials, sample projects, and community support are abundant.

- Standard Size: It's large enough to be easy to handle, and is compatible with most *shields* (additional boards that attach to the Arduino to add functionality).
- Sufficient Pin Count: Has sufficient I/O pin count for most beginner projects.
- 5V Operating Voltage: Many sensors and starter modules are designed to operate at 5V.
- Affordable Price: Relatively inexpensive, especially considering the clone version.

Another good alternative for beginners is the Arduino Nano, which has similar functionality to the UNO but in a much smaller size, making it suitable for mounting directly on a breadboard.

Introduction to Arduino IDE

The Arduino IDE (Integrated Development Environment) is free software that you install on your computer (Windows, macOS, or Linux) to write, compile, and upload program code to an Arduino board.

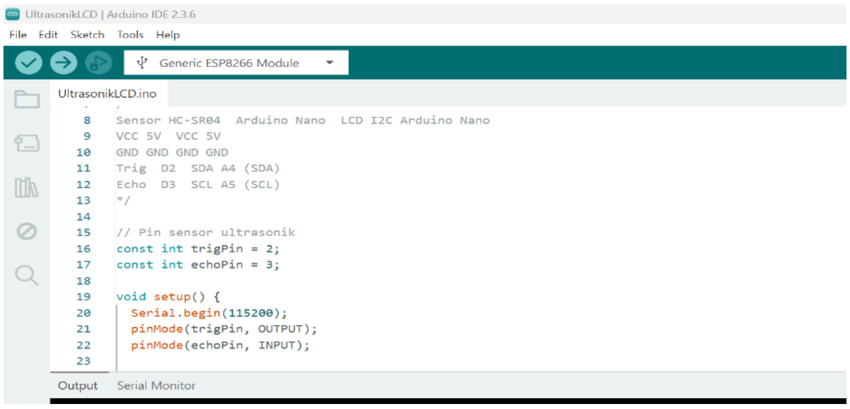
Installation:

- Download the latest version from the official Arduino website (www.arduino.cc).
- Follow the installation instructions for your operating system. This process will usually also install the USB drivers needed for your computer to recognize the Arduino board.

Main Interface:

- Code Area (Text Editor): Where you write your program code (called *sketch* in Arduino terminology).
- Verify button (✓): Compiles your code to check for syntax errors without uploading it to the board.
- Upload button (→): Compiles your code and, if there are no errors, uploads it to the connected Arduino board.
- Sketch Name: Displays the name of the currently open sketch file.
- Message Area: Displays status messages during compilation and upload, as well as error messages if any.
- Output Console: Displays more detailed output from the compilation and upload process.
- Board & Port menu (Tools > Board & Tools > Port): Where you select the type of Arduino board you are using and the serial COM port the board is connected to on your computer. Important: Make sure you select the correct board and port before uploading!

- Serial Monitor (Button in the top right corner or Tools > Serial Monitor): Opens a separate window that allows two-way communication between the Arduino and the computer via the USB/Serial connection. Very useful for displaying data from the Arduino (e.g. sensor values) or sending commands to the Arduino.



```
Sensor HC-SR04  Arduino Nano  LCD I2C Arduino Nano
VCC  5V   VCC  5V
GND  GND  GND  GND
Trig  D2   SDA  A4  (SDA)
Echo  D3   SCL  A5  (SCL)
*/

// Pin sensor ultrasonik
const int trigPin = 2;
const int echoPin = 3;

void setup() {
  Serial.begin(115200);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

```

Figure 6 Arduino IDE *(Arduino, 2025)*

With an understanding of the basic components of the board and how the IDE works, you are now ready to move on to the next chapter, where we will cover the electronics basics needed to start building simple circuits with your Arduino.

Reference:

- https://www.arduino.cc/en/hardware/
- https://fikom.udb.ac.id/artikel/detail/komponen-arduino-dan-fungsinya

---

Chapter 2: Basic Digital Electronics for Arduino

After getting to know the Arduino board and its IDE, the next step is to understand some basic electronic components that you will often use in Arduino projects. This chapter will discuss the basic concepts of electricity, an introduction to breadboards as a place to assemble, and the functions and workings of components such as resistors, LEDs, push buttons, and transistors.

A Brief Introduction to Current, Voltage, and Resistance (Ohm's Law)

Before assembling the components, it is important to understand three basic concepts in electricity:

1. Voltage (V): Can be thought of as the "push" or "pressure" that causes electrical charge (electrons) to flow. The unit of voltage is Volts (V). Arduino UNO typically operates at 5V.
2. Current (Current - I): Is the flow of electric charge itself. The unit of current is Ampere (A), but in Arduino projects often use smaller units such as milliamperes (mA) (1A = 1000mA).
3. Resistance (Resistance - R): Is the resistance to the flow of electric current. Components that have resistance are called resistors. The unit of resistance is Ohm (Ω). The greater the resistance, the more difficult it is for current to flow.

These three concepts are related to each other through Ohm's Law, which states:

`V = I * R`

That is, voltage (V) is proportional to the product of current (I) and resistance (R). This law is very important for calculating the correct component values, for example calculating the resistor needed for an LED.

Breadboard: A Place to Assemble Without Soldering

A breadboard (or often called a project board) is a plastic board with lots of small holes that allows you to assemble electronic components and connect them with jumper wires without soldering. It is ideal for prototyping and experimenting.

How Breadboard Works:
- Connected Holes: The holes on the breadboard are connected internally by metal clips.
- Terminal Row (Center): Holes in a short row (usually 5 holes) in the center of the breadboard are connected horizontally. Different rows are not connected.

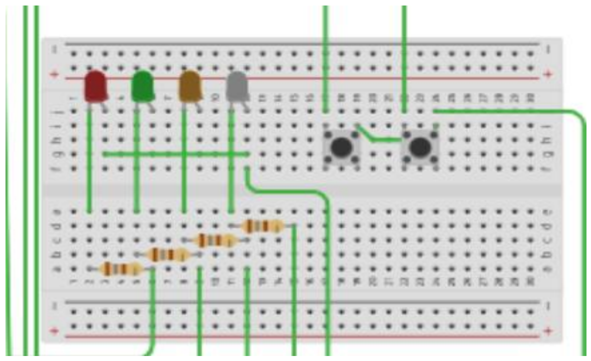


Figure 7 Breaboard *(ThinkerCad, 2025)*

Power Strips (Bus Strips - Edge Section): Usually consists of

- on two columns on each side of the breadboard (marked '+' and '-') whose holes are connected vertically along the columns. These are used to distribute power (VCC/+5V) and ground (GND/-) lines throughout the circuit.
- Central Notch (Divider): A gap in the middle that separates the two parts of the terminal row, suitable for installing ICs (Integrated Circuits) so that the pins on both sides are not connected to each other.

With a breadboard, you can easily mount components, connect them with jumper wires, and modify the circuit if necessary.

Resistor: The Current Resistor

A resistor is a passive component whose main function is to hold or limit the flow of electric current in a circuit. The value of its ability to hold current is called resistance, measured in Ohms (Ω).

Main Functions of Resistors in Arduino Projects:

1. Current Limiter: The most common function. For example, an LED requires a certain current to light up brightly but will be damaged if the current is too large. A resistor is placed in series with the LED to limit the current flowing through it according to Ohm's Law.
2. Pull-up / Pull-down Resistors: Used with input components such as buttons (push buttons) to ensure that the Arduino input pin always gets a definite HIGH or LOW signal when the button is not pressed, preventing *floating* conditions (will be discussed further in the Push Button section).

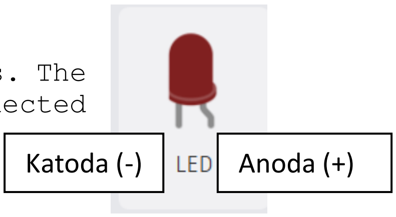
3. Voltage Divider: Two resistors connected in series can be used to step down the source voltage to a lower voltage level.

Reading Resistor Values (Color Code - Optional):
Physical resistors usually have colored bands that indicate their resistance value and tolerance. You can search for "resistor color code calculator" online to learn how to read them. However, for beginners, it's often easier to use a multimeter to measure resistance values or buy a resistor kit that already has the values labeled.

LED: The Light Giver

LED (Light Emitting Diode) is a semiconductor component that emits light when an electric current flows through it. LEDs are the most commonly used output component in beginner Arduino projects.

LED characteristics:
- Diode: Current can only flow in one direction.
- Anode (+) and Cathode (-) Legs: LEDs have two legs. The longer leg is the Anode (+), which should be connected to the positive voltage side (e.g. Arduino output pin or VCC). The shorter leg is the Cathode (-), which should be connected to the negative voltage side (Ground/GND).
- Forward Voltage (Vf): The minimum voltage required for the LED to start lighting (varies depending on color, usually around 1.8V - 3.3V).
- Forward Current (If): The ideal current for the LED to light up brightly and last a long time (usually around 10mA - 20mA).



Important: Using Current Limiting Resistors!
LEDs cannot be directly connected between the Arduino output pin and GND. The Arduino output pin (5V) is much higher than the LED Vf, and without a limiter, the current flowing will be very large and damage the LED (or even the Arduino pin). Therefore, always install a resistor in series with the LED. Common resistor values used for standard LEDs with Arduino 5V are between 220Ω to 1kΩ (eg 330Ω or 470Ω).

Resistor Calculation (Example):
- If V_source = 5V (from Arduino pin), Vf_LED = 2V, and If_LED = 15mA (0.015A), then:
- The voltage that needs to be "dissipated" by the resistor (Vr) = V_source - Vf_LED = 5V - 2V = 3V.
- Resistor Value (R) = Vr / If_LED = 3V / 0.015A = 200Ω. (Use the nearest larger standard value, e.g. 220Ω).

Push Button: Providing Simple Input

A push button is the simplest momentary switch. It connects the circuit when pressed and breaks it when released. This is an easy way for the user to provide input to the Arduino.

How it Works: When pressed, two internal contacts connect.

Problem: Floating Pin
If an Arduino input pin is not connected to VCC (HIGH) or GND (LOW) exactly (for example when a button is not pressed in a simple circuit), it is in a *floating* state. `digitalRead()` readings on this pin can be random (sometimes HIGH, sometimes LOW), making the program unreliable.

Solution: Pull-up or Pull-down circuit
To overcome floating, we use a resistor to "pull" the input pin to a certain voltage level when the button is off.

1. Pull-down sequence:
   a. The button connects the input pin to VCC (5V) when pressed.
   b. A resistor (e.g. 10kΩ) connects the input pin to GND.
   c. When not pressed: The pin is pulled LOW by the resistor.
   d. When pressed: Pin is connected to HIGH.
   e. Reading: LOW (not pressed), HIGH (pressed).
   f. `pinMode(pinButton, INPUT);`

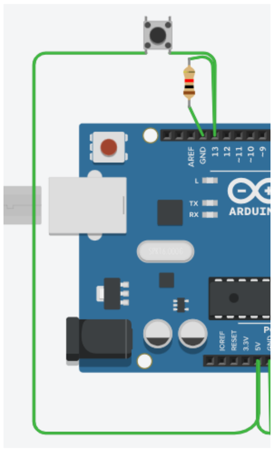


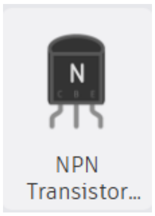
Figure 8 Pull-down circuit

2. Pull-up Circuit (Internal): The Easiest Way in Arduino
   a. The Arduino has internal pull-up resistors that can be enabled via code.
   b. The button connects the input pin to GND when pressed.
   c. No external resistors needed.
   d. When not pressed: The pin is pulled HIGH by the internal resistor.
   e. When pressed: Pin is connected to LOW.
   f. Reading: HIGH (not pressed), LOW (pressed).
   g. `pinMode(pinButton, INPUT_PULLUP);`

Due to its simplicity, using the internal pull-up resistor (`INPUT_PULLUP`) is highly recommended for connecting buttons to the Arduino.

Transistor: Powered Electronic Switch

Arduino output pins can only provide a limited amount of current (around 20mA-40mA per pin). This is enough to light an LED, but not enough to control devices that require more current, such as DC motors, relays, or long LED strips.



This is where transistors come in. Transistors are semiconductor components that can function as electronic switches or amplifiers.

Function as a Switch:
In the context of Arduino, we often use transistors (specifically NPN BJT types such as the 2N2222 or BC547, or MOSFETs) as switches. The Arduino output pins provide control signals (a small current to the Base of a BJT, or a voltage to the Gate of a MOSFET) to control the flow of a much larger current through the Collector-Emitter (BJT) or Drain-Source (MOSFET) path connected to a load (such as a motor).

Simple Working Method (NPN BJT as Switch):

1. The load (e.g. motor) is connected between the positive voltage source (which may be higher than 5V) and the Collector of the transistor.
2. The emitter of the transistor is connected to Ground (GND).
3. The base of the transistor is connected to the Arduino output pin through a base current limiting resistor (e.g. 1kΩ).
4. When the Arduino pin sends a HIGH signal, a small current flows into the Base, "opening" the path between the Collector and Emitter, allowing a large current to flow through the load (the spinning motor).
5. When the Arduino pin sends a LOW signal, there is no Base current, the Collector-Emitter path is "closed", and current to the load is cut off (the motor stops).

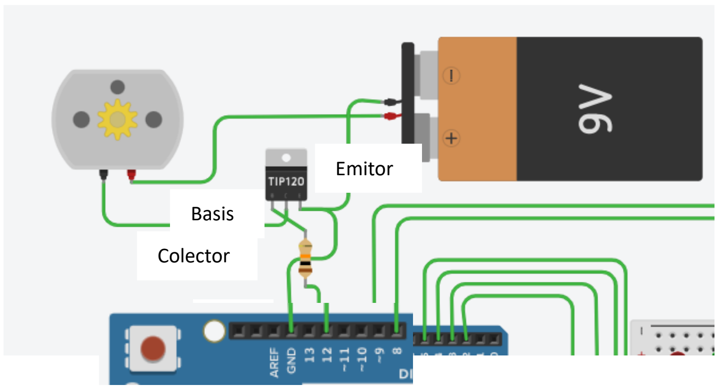


Figure 9 Transistor Circuit Controlling a Motor

***Important Note:* When using transistors to control inductive loads such as motors or relays, it is often necessary to use a *flyback* diode (placed in reverse parallel with the load) to protect the transistor from voltage spikes when the load is turned off.**

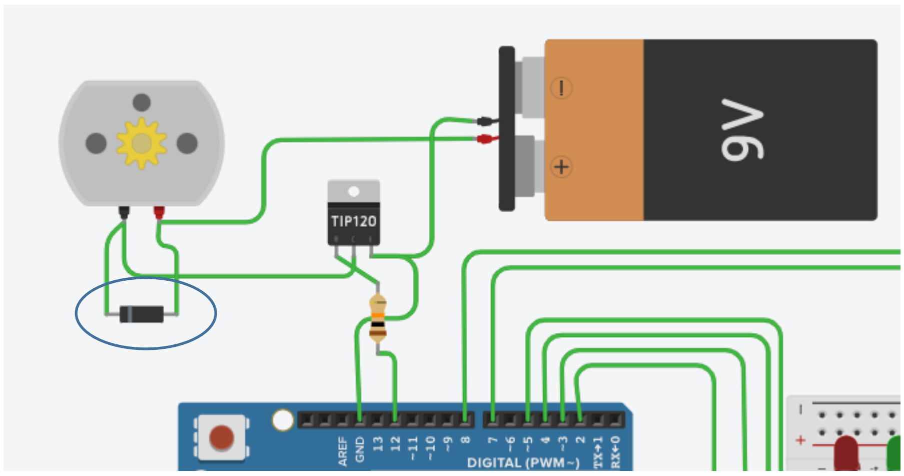


Figure 10 Transistor circuit controlling a motor equipped with a flyback diode

While the details of how transistors work can be complex, a basic understanding of them as Arduino-controlled switches is enough for many beginner projects.
With an understanding of these basic components, you are ready to begin learning how to program the Arduino to interact with them in the next chapter.

Reference:

- https://www.kompas.com/skola/read/2022/06/21/103000269/fungsi-resistor-pada-rangkaian-elektronika
- https://www.arduino.biz.id/2021/01/pengertian-transistor-jenis-fungsi-dan.html
- https://medium.com/@arisadityanugraha/menggunakan-push-button-pada-arduino-a149337288ff

Chapter 3: Getting to Know C Language & Basic Algorithms

Now that we understand the Arduino hardware and the basics of electronics, it's time to dive into the heart of Arduino control: programming. Arduino is programmed using a language based on C/C++, a powerful and widely used programming language. This chapter will introduce you to the basic structure of an Arduino program, how to use variables, operators, important functions for controlling pins, basic algorithm concepts (sequences, branches, loops), how to communicate with a computer via a serial port, and non-blocking scheduling techniques using `millis()`.

Basic Structure of Arduino Program

Every Arduino program, or commonly called *sketch*, has a basic structure that must be present. At a minimum, a sketch consists of two main functions: `setup()` and `loop()`. In addition, there is usually a section for declaring global variables.

1. Global Variable Declaration:
   a. Variables declared outside of all functions (usually at the very top of the sketch).
   b. Can be recognized and used by all functions in sketch (`setup()`, `loop()`, and custom functions).
   Example: `const int pinLed = 13;`

3. `setup()` function:
   a. Must have.
   b. Runs only once when the Arduino is first powered on or after the reset button is pressed.
   c. Used to perform initialization or initial setup required before the main program runs. For example:
   d. Set the pin mode using **`pinMode()`**.
   e. Start serial communication with ` **Serial.begin()**`.
   f. Initialize external libraries.

```cpp
void setup() {
     // Initialization code here
     pinMode(pinLed, OUTPUT); // Example: set LED pin as
     output
     Serial.begin(9600); // Example: starting serial
     communication
}
```

4. The `loop()` function:
   a. Must have.
   b. Executed continuously and repeatedly after `setup()` completes.
   c. This is the core of the program where your main logic runs. Arduino will execute the code inside the `loop()` from top to bottom, then back to the top, and repeat this endlessly.

~~~
       ```cpp
       void loop() {
                 // The main code of the program is here, it will be
                 repeated continuously
                 digitalWrite(pinLed, HIGH); // Example: turn on the LED
                 delay(500); // Example: 500ms delay
                 digitalWrite(pinLed, LOW); // Example: turn off the LED
                 delay(500); // Example: 500ms delay
           }
           ```
~~~

Example of turning on the light based on the button

```
// C++ code
//
const int LedR=2;
const int LedG=3;
const int LedO=4;
const int LedW=5;
const int btn1=7;
const int btn2=8;

void setup()
{
for (int a=LedR ; a<=LedW ; a++)
pinMode(a, OUTPUT);
for (int a=btn1 ; a<=btn2 ; a++)
pinMode(a, INPUT_PULLUP);
for (int a=LedR ; a<=LedW ; a++)
digitalWrite(a, HIGH);
initLed();
}

void initLed()
{
for (int a=LedR ; a<=LedW ; a++)
{digitalWrite(a, LOW);
delay(500);
}
for (int a=LedW ; a>=LedR ; a--)
{digitalWrite(a, HIGH);
delay(500);
}


}
void loop()
{
int readbtn1=digitalRead(btn1);
int readbtn2=digitalRead(btn2);
if (readbtn1==LOW)
{
for (int a=LedR ; a<=LedW ; a++)
{digitalWrite(a, LOW);
```

```
delay(500);
digitalWrite(a,HIGH);
}
}
if (readbtn2==LOW)
{
for (int a=LedW ; a>=LedR ; a--)
{digitalWrite(a, LOW);
delay(500);
digitalWrite(a,HIGH);
}
}

}
```

Variables and Data Types

A variable is a container for storing data that can change during the program's execution. Each variable must have a data type that determines what kind of value it can store.

Common Data Types in Arduino:

- `int`: Stores an integer (no decimals), positive or negative. Example: `int numberOfStudents = 25;`
- `float`: Stores decimal (fractional) numbers. Example: `float temperature = 27.5;`
- `boolean`: Stores the logical value `true` or `false`. Example: `boolean lampuNyala = true;`
- `char`: Stores a single character (enclosed in single quotes). Example: `char option = 'A';`
- `String`: Stores a sequence of characters (text) (enclosed in double quotes). More flexible but uses more memory than a char array. Example: `String message = "Hello Arduino!";`
- `byte`: Stores small positive integers (0 to 255).
- `long`: Stores integers with a range greater than `int`.
- `unsigned long`: Stores positive integers with a very large range. This data type is returned by the `millis()` function.

Variable Declaration:

`DataTypeVariableName;` or `DataTypeVariableName = InitialValue;`
Example: `int counter = 0;`

Local vs Global Variables:

- Global: Declared outside the function, can be accessed from anywhere.
- Local: Declared inside a function, can only be accessed inside that function.

Operator

Operators are used to perform operations on variables and values.

- Arithmetic Operators: `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulo/remainder).

- Comparison Operators: `==` (equal to), `!=` (not equal to), `<` (less than), `>` (greater than), `<=` (less than or equal to), `>=` (greater than or equal to). The result is a `boolean` (`true` or `false`).
- Logical Operators: `&&` (AND - `true` if both sides are `true`), `||` (OR - `true` if either side is `true`), `!` (NOT - inverts a `boolean` value).
- Assignment Operators: `=` (assign value), `+=`, `-=`, `*=`, `/=` (operation and assignment).
- Increment/Decrement Operators: `++` (plus 1), `--` (minus 1).

Basic Input/Output (I/O) Functions

These are the core functions for interacting with Arduino pins.

- **`pinMode(pin, mode)`**:
  - Sets the function of a digital pin.
  - `pin`: The pin number to set.
  - `mode`: Can be `INPUT` (to read a signal), `OUTPUT` (to send a signal), or `INPUT_PULLUP` (input with internal pull-up resistor active).
  - Usually called inside `setup()`.
  - Example: `pinMode(13, OUTPUT);`

- **` digitalWrite(pin, value)`**:
  - Writes a digital value (HIGH or LOW) to a pin that is set as `OUTPUT`.
  - `pin`: Pin number.
  - `value`: `HIGH` (usually 5V or 3.3V) or `LOW` (0V/GND).
  - Example: `digitalWrite(13, HIGH);` // Turn on the LED on pin 13

- **`digitalRead(pin)`**:
  - Reads a digital value (HIGH or LOW) from a pin that is set as `INPUT` or `INPUT_PULLUP`.
  - `pin`: Pin number.
  - Returns `HIGH` or `LOW`.
  - Example: `int statusButton = digitalRead(2);`

- **`analogRead(pin)`**:
  - Read analog values from analog input pins (A0-A5 on UNO).
  - `pin`: Analog pin number (eg: `A0`).
  - Returns an integer value between 0 (for 0V) and 1023 (for 5V or AREF reference voltage).
  - Example: `int SensorValue = analogRead(A0);`

- **` analogWrite(pin, value)`**:
  - Writes "pseudo" analog values (PWM - Pulse Width Modulation) to digital pins that support PWM (marked '~' on the UNO, e.g. pins 3, 5, 6, 9, 10, 11).
  - `pin`: PWM pin number.
  - `value`: A value between 0 (always OFF) and 255 (always ON). Values in between produce a PWM signal with varying duty cycle.

- o Useful for controlling LED brightness or DC motor speed.
- o Example: `analogWrite(9, 128);` // Sets PWM output pin 9 to 50% duty cycle

Basic Algorithm

An algorithm is a sequence of logical steps to solve a problem. In programming, we use control structures to implement algorithms.

1. Sequential: Instructions are executed one at a time, line by line. This is the basic flow in `setup()` and `loop()`.

2. Branching (Selection): Selecting an execution path based on conditions.
   a. `if (condition) { ... }`: Execute a block of code if the condition is `true`.
   b. `if (condition) { ... } else { ... }`: Execute the first block if `true`, the second block if `false`.
   c. `if (condition1) { ... } else if (condition2) { ... } else { ... }`: Checks multiple conditions in sequence.
   d. `switch(variable) { case value1: ... break; case value2: ... break; default: ... }`: Compares a variable with some constant value.
   ```cpp
   int temperature = analogRead(A0);
   if (temperature > 500) {
   Serial.println("Hot!");
   } else {
   Serial.println("Normal");
   }
   ```

3. Looping: Running a block of code repeatedly.
   a. `for (initialize; condition; iteration) { ... }`: Usually for loops with a known number of iterations.
   b. `while (condition) { ... }`: Checks condition before iteration, runs as long as condition is `true`.
   c. `do { ... } while (condition);`: Runs code at least once, then checks condition.
   ```cpp
   for (int i = 0; i < 5; i++) {
   Serial.print("Iterate to: ");
   Serial.println(i);
   }
   ```

Function/Sub-program

You can create your own functions to group code that performs specific tasks. This makes the program more structured, easier to read, and the code can be reused.

```cpp
// Function declaration (can be above or below setup/loop)
void blinkLed(int pin, int duration) {
digitalWrite(pin, HIGH);
delay(duration);
digitalWrite(pin, LOW);
delay(duration);
}

void setup() {
pinMode(13, OUTPUT);
}

void loop() {
blinkLed(13, 250); // Calling custom function}
```

Serial Communication

How Arduino communicates with a computer (via Serial Monitor) or other devices.

- `Serial.begin(baudRate)`: Starts serial communication (usually in `setup()`). Common `baudRate` is 9600.
- `Serial.print(data)`: Sends data as text to the serial port without newlines.
- `Serial.println(data)`: Sends data as text, followed by a newline character.
- `Serial.available()`: Checks if there is incoming data available to read. Returns the number of bytes available.
- `Serial.read()`: Reads one byte of incoming data from the serial buffer. Returns the byte value (0-255) or -1 if no data.

```cpp
void setup() {
Serial.begin(9600);
}

void loop() {
if (Serial.available() > 0) {
char dataIn = Serial.read(); // Read one character
Serial.print("You sent: ");
Serial.println(dataInput);
}
}
```

`millis()` Timer Concept (Non-Blocking)

The `delay()` function is very easy to use but has a major drawback: it blocks or stops the execution of the program altogether for a specified duration. While `delay()` is active, the Arduino cannot do anything else.

The `millis()` function is a solution for pausing or scheduling without blocking. It returns the number of milliseconds since the Arduino started running (as an `unsigned long`).

Common Usage Patterns for `millis()`:

1. Store the last time an action was performed (`previousMillis`).
2. Inside `loop()`, get the current time (`currentMillis = millis();`).
3. Check if the difference between the current time and the last time has reached the desired interval (`if (currentMillis - previousMillis >= interval)`).
4. If yes, take action and immediately update `previousMillis = currentMillis;`.
5. Other code inside `loop()` can continue to run uninterrupted.

~~~~
```cpp
unsigned long previousMillis = 0;
const long interval = 1000; // 1 second interval
int ledState = LOW;

void setup() {
pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
unsigned long currentMillis = millis();

// LED scheduling section
if (currentMillis - previousMillis >= interval) {
previousMillis = currentMillis; // Last time update
// Flip the LED status
ledState = !ledState;
digitalWrite(LED_BUILTIN, ledState);
}

// Arduino can do other tasks here at the same time
// For example, reading a sensor or a button
}
```
~~~~

This understanding of basic C language, algorithmic structures, I/O functions, serial communication, and `millis()` will be a strong foundation for you to build more complex Arduino projects.

Reference:

- https://mikroavr.com/struktur-dasar-program-arduino/
- https://mamikos.com/info/struktur-dasar-algoritma-pljr/
- https://www.arduino.cc/reference/tr/language/functions/communication/serial/print
- https://docs.arduino.cc/language-reference/en/functions/communication/serial/read/

- https://www.norwegiancreations.com/2017/09/arduino-tutorial-using-millis-instead-of-delay/

Chapter 4: Basic Implementation: LED & Button Control

Now that you've learned the basics of electronics and Arduino programming, it's time to combine that knowledge to create simple interactions. This chapter will walk you through the most basic implementations: controlling an LED output and reading input from a push button. We'll start with a hardware version of the "Hello, World!" program, blinking an LED, then learn to read button input, and combine it to control an LED with a button.

Blink: Make an LED Blink (Using `delay()`)

This is the first program that Arduino beginners usually try. The goal is simple: to make an LED turn on and off alternately.

Required Components:
- Arduino Board (UNO, Nano, etc.)
- USB cable
- (Optional, if not using the built-in LED) 1x LED, 1x Resistor 220Ω - 1kΩ, Jumper Cable, Breadboard.

Series:
The easiest way is to use the built-in LED on the Arduino board. This LED is usually connected to digital pin 13 (on the UNO/Nano). Arduino provides a constant `LED_BUILTIN` that refers to this pin, making the code more portable.

If using an external LED:
1. Connect the short leg of the LED (Cathode) to the Arduino GND.
2. Connect the long leg of the LED (Anode) to one end of the resistor (eg 330Ω).
3. Connect the other end of the resistor to the `LED_BUILTIN` digital pin (or pin 13 on the UNO).

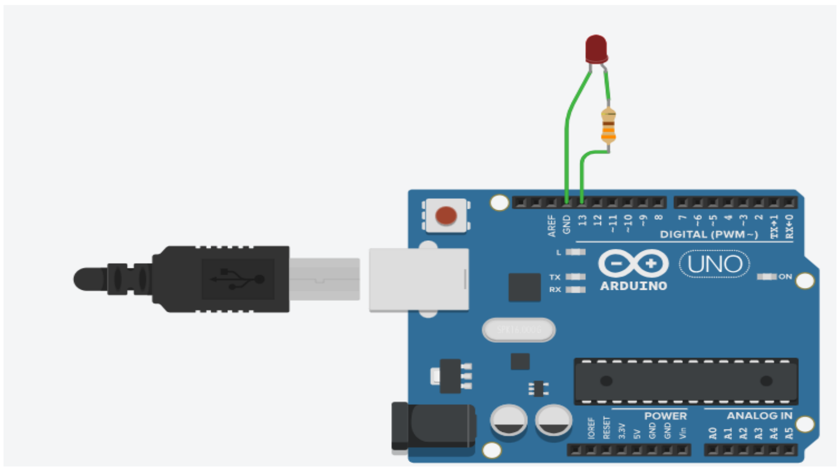


Figure 11 Wiring LED Flashing

~~~
        Program Code:
        ```cpp
        // Basic Blink Program

        void setup() {
        // Set the LED_BUILTIN pin as an output
        pinMode(LED_BUILTIN, OUTPUT);
        }

        void loop() {
        digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED
        delay(1000); // Pause for 1000 milliseconds (1 second)
        digitalWrite(LED_BUILTIN, LOW); // Turn off the LED
        delay(1000); // Pause for 1000 milliseconds (1 second)
        }
```
~~~

Ways of working:

- `setup()`: Sets the `LED_BUILTIN` pin as `OUTPUT`.
  `loop()`:
- `digitalWrite(LED_BUILTIN, HIGH);`: Apply voltage to the pin, LED lights up.
- `delay(1000);`: The program stops completely for 1 second.
- `digitalWrite(LED_BUILTIN, LOW);`: Stops voltage to the pin, LED turns off.
- `delay(1000);`: The program stops completely again for 1 second.
- The process inside this `loop()` keeps repeating itself.

Blink Without Delay: Blinking an LED Using `millis()`

As discussed in Chapter 3, `delay()` blocks the program. If we want the Arduino to do something else while the LED is blinking (for example reading a sensor), we need to use `millis()` using the Wiring in Figure 11 .

~~~
        Program Code (Blink Without Delay):
        ```cpp
        // Blink Program Without Delay

        const int ledPin = LED_BUILTIN; // LED pins
        int ledState = LOW; // Stores the current LED state (LOW or HIGH)

        unsigned long previousMillis = 0; // Stores the last time the LED
        changed state
        const long interval = 1000; // Blink interval (1000 ms = 1 second)

        void setup() {
        pinMode(ledPin, OUTPUT);
        }

        void loop() {
        // Get the current time
        unsigned long currentMillis = millis();

        // Check if it's time to change the LED status
~~~

```
if (currentMillis - previousMillis >= interval) {
// Save the current time as the last change time
previousMillis = currentMillis;

// Change the LED status: if it is LOW it will be HIGH, if it is
HIGH it will be LOW
if (ledState == LOW) {
ledState = HIGH;
} else {
ledState = LOW;
}

// Apply new state to LED
digitalWrite(ledPin, ledState);
}

// The program can do other things here without being disturbed by
the LED blinking.
// For example: Serial.println("Arduino is working...");
}
```

Ways of working:

- The program does not stop. In each iteration of `loop()`, it checks whether 1 second has passed since the last LED state change.
- If so, it changes the LED status (`HIGH` to `LOW` or vice versa) and records the time of the change.
- This allows the Arduino to remain responsive to other tasks.

Reading Button (Push Button)

Now we will learn to read input from the user through buttons.

Required Components:

- Arduino Board
- USB Cable
- 1x Push Button
- (Optional, if not using internal pull-up) 1x 10kΩ Resistor
- Jumper Cable
- Breadboard

Circuit (Using Internal Pull-up - Recommended):

1. Mount the buttons on the breadboard.
2. Connect one of the button legs to an Arduino digital pin (e.g. pin 2).
3. Connect the *opposite* button leg (the one that is internally connected when pressed) to the Arduino's GND.
4. No external resistors needed.

Program Code (Reading Buttons with Internal Pull-up):

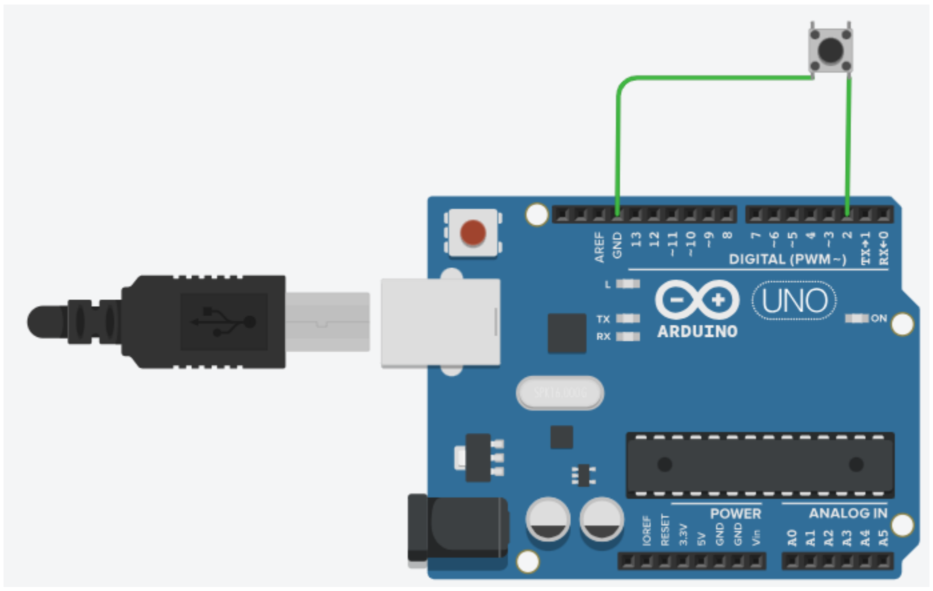


Figure 12 Push Button Circuit

```cpp
// Button Reading Program (Internal Pull-up)

const int buttonPin = 2; // Button pin

void setup() {
Serial.begin(9600); // Start serial communication to see results
// Set the button pin as INPUT with the internal pull-up resistor
active.
pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
// Read button status
int buttonState = digitalRead(buttonPin);

// Display status to Serial Monitor
// (LOW if pressed, HIGH if not pressed due to pull-up)
Serial.println(buttonState);

delay(50); // Give a little pause for reading stability
}
```

Ways of working:

- `pinMode(buttonPin, INPUT_PULLUP);`: Sets pin 2 as an input and enables the internal pull-up resistor. This makes pin 2 read HIGH when the button is not pressed.
- `digitalRead(buttonPin);`: Read the status of pin 2.
- When the button is not pressed, pin 2 is pulled HIGH by the internal resistor, `digitalRead()` returns `HIGH` (1).
- When the button is pressed, pin 2 is connected to GND, `digitalRead()` returns `LOW` (0).
- The result (1 or 0) is sent to the Serial Monitor.

Controlling LED with Buttons

Let's combine the two previous examples: turning on an LED only when a button is pressed.

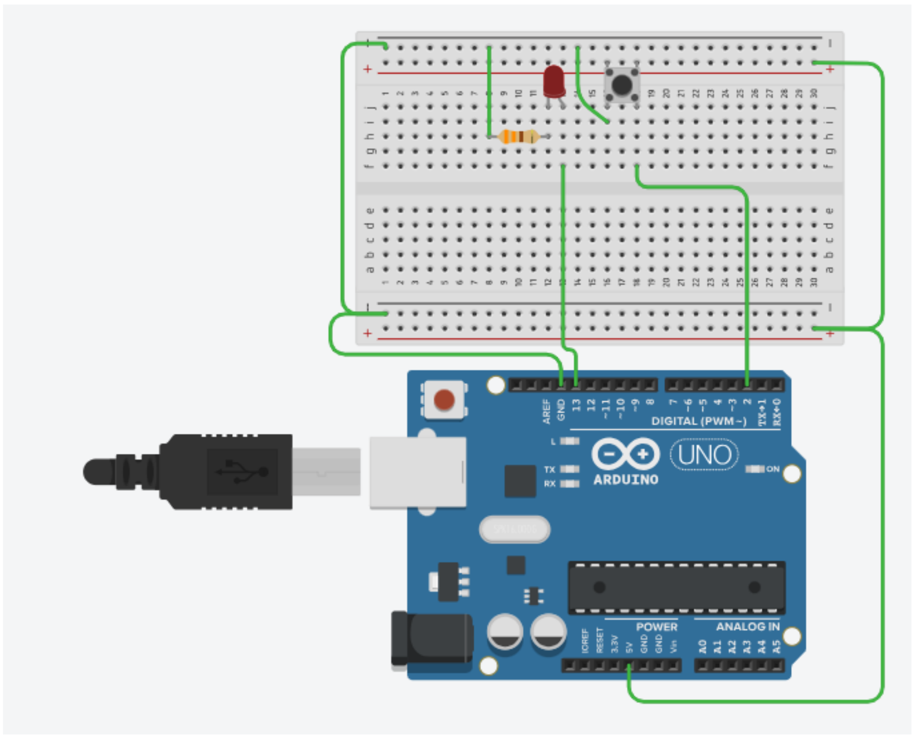


Figure 13 Combination of the two examples above, LED and Push Button

Circuit: Same as the button reading circuit (internal pull-up), plus LED (can be the built-in LED `LED_BUILTIN` or external LED with a resistor on pin 13).

Program Code:
```cpp
// LED Control Program with Button (Internal Pull-up)

const int buttonPin = 2; // Button pin
const int ledPin = LED_BUILTIN; // LED pins

void setup() {
pinMode(ledPin, OUTPUT);
pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
// Read button status
int buttonState = digitalRead(buttonPin);

// Check if button is pressed (LOW due to pull-up)
if (buttonState == LOW) {
```

~~~
// If pressed, turn on the LED
digitalWrite(ledPin, HIGH);
} else {
// If not pressed, turn off the LED
digitalWrite(ledPin, LOW);
}
}
```
~~~

Ways of working:

- The program continuously reads the button status inside `loop()`.
- If `digitalRead()` returns `LOW` (meaning the button is pressed), then `digitalWrite()` is used to turn on the LED (`HIGH`).
- If `digitalRead()` returns `HIGH` (button not pressed), then the LED is off (`LOW`).

Other Cases (Development Examples)

1. Toggle LED: Makes the LED state change (ON to OFF, OFF to ON) every time the button is pressed. This requires additional logic to detect the *change* in button state (from not pressed to pressed) and a variable to store the last LED state.

~~~
```cpp
// Example of Toggle Logic (needs to be refined with debouncing)
int ledState = LOW;
int lastButtonState = HIGH; // Assume the button was not pressed
initially
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;

void loop() {
int reading = digitalRead(buttonPin);
unsigned long currentMillis = millis();

if (reading != lastButtonState) {
lastDebounceTime = currentMillis;
}

if ((currentMillis - lastDebounceTime) > debounceDelay) {
if (reading != buttonState) {
buttonState = reading;
if (buttonState == LOW) { // Button was just pressed
ledState = !ledState; // Flip the LED state
digitalWrite(ledPin, ledState);
}
}
}
lastButtonState = reading;
}
```
~~~

Note: The toggle code above includes the basic concept of "debouncing" using `millis()` to deal with double reading when the button is pressed/released.

2. Controlling Multiple LEDs: You can use multiple buttons to control multiple LEDs independently, or one button to control the flashing pattern of multiple LEDs.

Experimenting with these examples will strengthen your understanding of digital inputs and outputs, as well as basic programming logic in Arduino.

Reference:

- https://docs.arduino.cc/built-in-examples/basics/Blink/
- https://docs.arduino.cc/built-in-examples/digital/Button/
- https://www.norwegiancreations.com/2017/09/arduino-tutorial-using-millis-instead-of-delay/

Chapter 5: Reading Sensors & Displaying to LCD

Once you are able to control basic outputs such as LEDs and read simple inputs from buttons, the next step is to make the Arduino interact further with its environment using sensors and display more complex information using an LCD (Liquid Crystal Display).

This chapter will introduce you to some popular sensors that are easy for beginners to use: a temperature and humidity sensor (DHT11), a light sensor (LDR/Photoresistor), and an ultrasonic distance sensor (HC-SR04). We'll also learn how to display data from these sensors (or any other) to a 16x2 character LCD display using a pin-saving I2C interface.

Sensors: Arduino Eyes and Ears

A sensor is a device that detects changes or events in the physical environment and converts them into electrical signals that can be read by a microcontroller such as Arduino. There are different types of sensors to measure different physical quantities.

1. Temperature and Humidity Sensor (DHT11)

The DHT11 is a popular low-cost digital sensor for measuring air temperature and relative humidity (RH).

- How it Works: Uses a capacitive humidity sensor and a thermistor to measure ambient air conditions. A chip inside processes the readings and sends digital data through a single data pin.
- General Specifications:
  1. Temperature Range: 0-50°C (Accuracy ±2°C)
  2. Humidity Range: 20-90%RH (Accuracy ±5%)
  3. Voltage: 3-5.5V
- Types: There are 4 pin (requires external 10kΩ pull-up resistor between VCC and Data) and 3 pin modules (usually includes pull-up resistor).
- Connection (3 Pin Module):
  1. VCC/+: to 5V Arduino
  2. GND/-: to Arduino GND
  3. Data/OUT/S: to Arduino Digital Pin (e.g. pin 2)
- Library: Requires a specific library. Adafruit's "DHT sensor library" is a common choice (install via Library Manager, may need "Adafruit Unified Sensor" as well).

DHT11 Reading Code Example:

~~~
```cpp
include "DHT.h" // Include the library

define DHTPIN 2 // Sensor data pin
define DHTTYPE DHT11 // Sensor type

DHT dht(DHTPIN, DHTTYPE); // Create a sensor object

void setup() {
Serial.begin(9600);
~~~

```
dht.begin(); // Initialize sensor
}

void loop() {
delay(2000); // DHT11 needs a delay between readings

float humidity = dht.readHumidity();
float temperature = dht.readTemperature(); // Celsius

// Check if reading failed
if (isnan(humidity) || isnan(temperature)) {
Serial.println("Failed to read DHT11!");
return;
}

Serial.print("Humidity: ");
Serial.print(humidity);
Serial.print(" %\t"); // \t = tabs
Serial.print("Temperature: ");
Serial.print(temperature);
Serial.println(" *C");
}
```

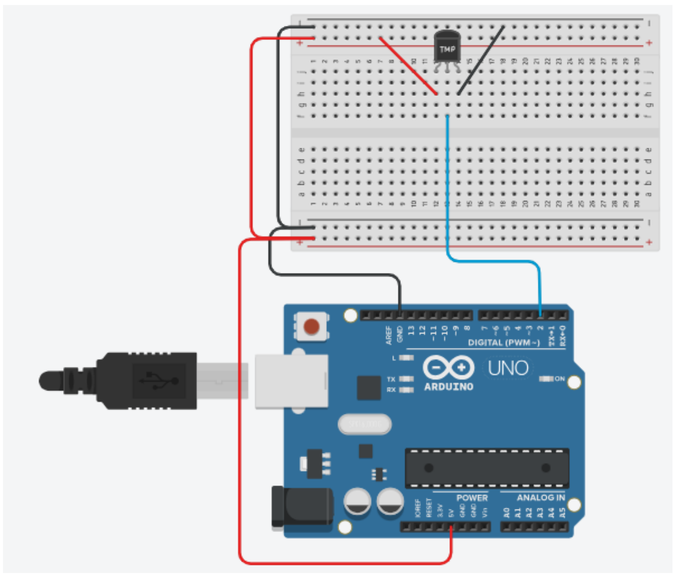


Figure 14 Heat sensor circuit

2. Light Sensor (LDR/Photoresistor)

LDR (Light-Dependent Resistor) is a resistor whose resistance value changes depending on the intensity of the light it receives: the brighter it is, the smaller the resistance.

- How it Works with Arduino: Since Arduino reads analog voltages, the LDR is used in a voltage divider circuit with a fixed resistor (e.g. 10kΩ). The change in resistance of the LDR will cause a change in voltage at the midpoint of the voltage divider, which is then read by the Arduino analog pin.

- General Voltage Divider Circuit:
  1. One leg of LDR to 5V.
  2. The other leg of the LDR goes to the Analog Pin (eg A0).
  3. The same Analog pin (A0) is connected to GND through a 10kΩ resistor.

Result: The voltage at A0 will be high when it is bright, low when it is dark.

- Reading: Use `analogRead(pinAnalog)` which returns a value of 0-1023 (proportional to voltage 0-5V).

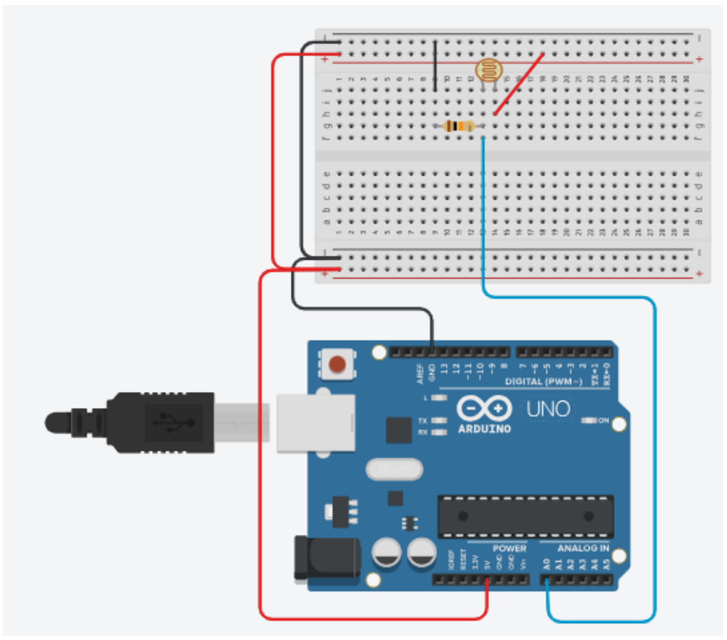


Figure 15 Light Sensor Circuit

LDR Reading Code Example:

```cpp
const int ldrPin = A0; // LDR analog pin

void setup() {
Serial.begin(9600);
}

void loop() {
int ldrValue = analogRead(ldrPin);

Serial.print("Light Sensor Value: ");
Serial.println(ldrValue); // Value 0-1023 (high when bright, low when
dark)

// Example: Turn on LED if dark (value < 300, need experimentation)
// if (ldrValue < 300) { digitalWrite(LED_BUILTIN, HIGH); }
// else { digitalWrite(LED_BUILTIN, LOW); }

delay(100);
}
```

3. Ultrasonic Distance Sensor (HC-SR04)

This sensor measures the distance to objects in front of it using ultrasonic sound waves, much like a bat's sonar.

- Ways of working:
  1. The Arduino sends a short pulse to the sensor's `Trig` pin.
  2. The sensor emits ultrasonic waves.
  3. The sensor makes the `Echo` pin HIGH.
  4. The waves bounce off the object and return to the sensor.
  5. When a reflection is received, the sensor makes the `Echo` pin go LOW.
  6. Arduino measures the duration of the `Echo` pin HIGH using `pulseIn()`.
  7. Distance is calculated from the duration (`Distance = (Duration * Speed of Sound) / 2`).
- General Specifications:
  1. Distance: 2cm - 400cm
  2. Voltage: 5V
- Connection:
  - VCC: to 5V Arduino
  - GND: to Arduino GND
  - Trig: to Arduino Digital Pin (eg 9, set to OUTPUT)
  - Echo: to Arduino Digital Pin (eg 10, set by INPUT)

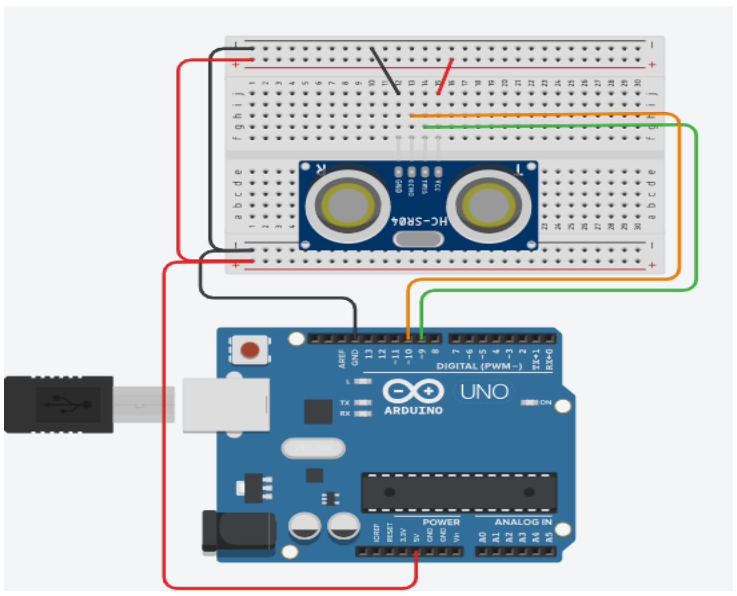


Figure 16 Ultrasonic Sensor Circuit

Example of Reading Code HC-SR04:
```cpp
const int trigPin = 9;
const int echoPin = 10;

long duration;
int distance; // Distance in cm

void setup() {
```

~~~
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
Serial.begin(9600);
}

void loop() {
// Send a 10 microsecond trigger pulse
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

// Read echo pulse duration
duration = pulseIn(echoPin, HIGH);

// Calculate the distance (speed of sound = 0.034 cm/µs)
distance = duration * 0.034 / 2;

// Show distance
Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");

delay(100);
}
```
~~~

Displaying Data to 16x2 LCD (via I2C)

A 16x2 (16 columns, 2 rows) character LCD is a popular way to display text information or sensor data without the need for a computer.

- Parallel Connection Problem: Connecting a standard LCD directly requires a lot of Arduino pins (minimum 6 digital pins).
- Solution: I2C LCD Adapter Module: A small module that is mounted behind the LCD, allowing control via the I2C protocol with only 2 data pins (SDA, SCL) plus VCC and GND.
- I2C advantages: Save pins, simple wiring, I2C bus can be shared with other I2C devices.
- Connection to Arduino UNO:
  - Module GND: to Arduino GND
  - VCC Module: to 5V Arduino
  - SDA Module: to Arduino Pin A4
  - SCL Module: to Arduino Pin A5
- I2C Address: Each I2C device has a unique address. LCD modules are usually `0x27` or `0x3F`. Use the *I2C Scanner Sketch* (File > Examples > Wire > i2c_scanner) to check if the LCD is not displaying anything.
- Contrast: Turn the blue potentiometer on the back of the I2C module to adjust the display contrast.
- Library: Requires `Wire.h` (standard) and `LiquidCrystal_I2C.h` libraries (install via Library Manager).

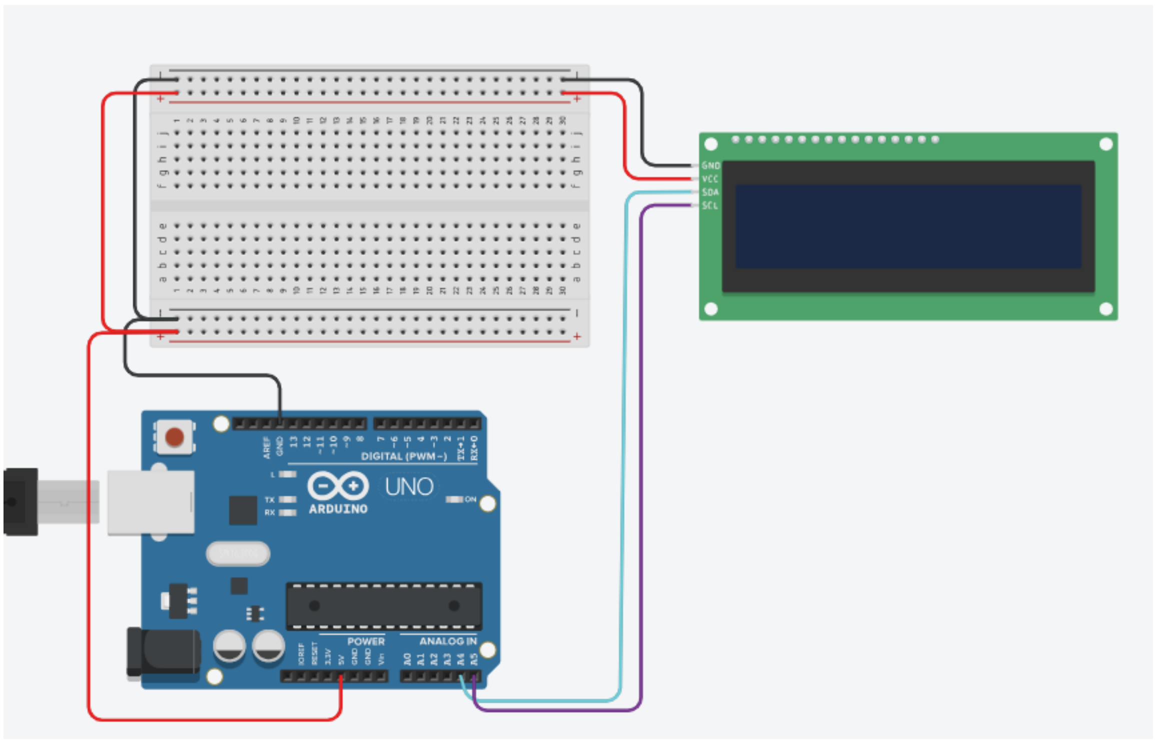


Figure 17 I2C LCD Circuit

~~~
Example Code for Displaying to I2C LCD:
```cpp
include <Wire.h>
include <LiquidCrystal_I2C.h>

// Replace 0x27 with your LCD I2C address if different
LiquidCrystal_I2C lcd(0x27, 16, 2); // Address, Column, Row

void setup() {
lcd.init(); // LCD initialization
lcd.backlight(); // Turn on the backlight

lcd.setCursor(0, 0); // Column 0, Row 0
lcd.print("Temperature:");
lcd.setCursor(0, 1); // Column 0, Row 1
lcd.print("Moist:");
}

void loop() {
// Suppose we have temperature and humidity variables
float temperature = 28.5; // Sample data
float damp = 65.2; // Sample data

// Display data to LCD
lcd.setCursor(7, 0); // Column 7, Row 0
lcd.print(temperature); // Display the temperature value
lcd.print(" C");

lcd.setCursor(7, 1); // Column 7, Row 1
lcd.print(humid); // Display humidity value
~~~

```
lcd.print(" %");

delay(1000); // Pause before next update
}
```

Important LCD Functions:

- `lcd.init()` or `lcd.begin()`: Initialization.
- `lcd.backlight()` / `lcd.noBacklight()`: Backlight control.
- `lcd.setCursor(column, row)`: Move cursor (starting from 0).
- `lcd.print(data)`: Display text or variable value.
- `lcd.clear()`: Clear the entire screen display.

With the ability to read various types of sensors and display the results on the LCD, you can start building more informative and interactive projects, such as a mini weather station, a digital distance meter, or a simple automation system based on environmental conditions.

Reference:

- https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/
- https://www.instructables.com/How-to-use-a-photoresistor-or-photocell-Arduino-Tu/
- https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
- https://lastminuteengineers.com/i2c-lcd-arduino-tutorial/

CHAPTER 6 Arduino Simulation with Tinkercad

Introduction to Tinkercad

Tinkercad is a web-based platform provided by Autodesk for 3D design, electronic circuit simulation, and microcontroller programming (such as Arduino). Tinkercad's advantages include:

- Free and online based (no installation required).
- User-friendly interface, suitable for students and beginners.
- Supports circuit simulation and direct Arduino code upload.
- Has a complete collection of components such as LEDs, servo motors, sensors, and others.

Website address: https://www.tinkercad.com/

Steps to Use Tinkercad for Arduino Simulation

Step 1: Registration and Login

- Open Tinkercad .
- Register a free account or login using your Google/Autodesk account.

Step 2: Create a New Project

- Click “Create New Circuit” on the dashboard.
- The circuit work window will open.

Step 3: Adding Components

- Use the sidebar on the right to search for and add components such as:
    - Arduino Uno
    - Servo motor
    - Potentiometer
    - Breadboard (if needed)
    - Resistors, cables, etc.

Step 4: Connecting the Components

- Use click-drag to connect cables between components.
- Use different colored cables to differentiate functions (eg: red = power, black = ground, blue = signal).

Step 5: Writing Arduino Code

- Click the “Code” tab.
- Use “Text” mode to write Arduino C code.
- Click “Start Simulation” to run the simulation.

Case Study: Servo Motor Controller Using Potentiometer

Objective

Create an Arduino simulation that drives **a servo motor based on potentiometer** values using Tinkercad.

Components Used

- 1 x Arduino Uno
- 1 x Servo motor
- 1 x Potentiometer
- Jumper cables
- (Optional) Breadboard

Series

- **Potentiometer** :
  - Middle leg → A0 (Arduino)
  - Left foot → 5V
  - Right foot → GND
- **Servo Motor** :
  - Signal cable (yellow/orange) → Pin 9 (Arduino)
  - VCC (red) → 5V
  - GND (black/brown) → GND

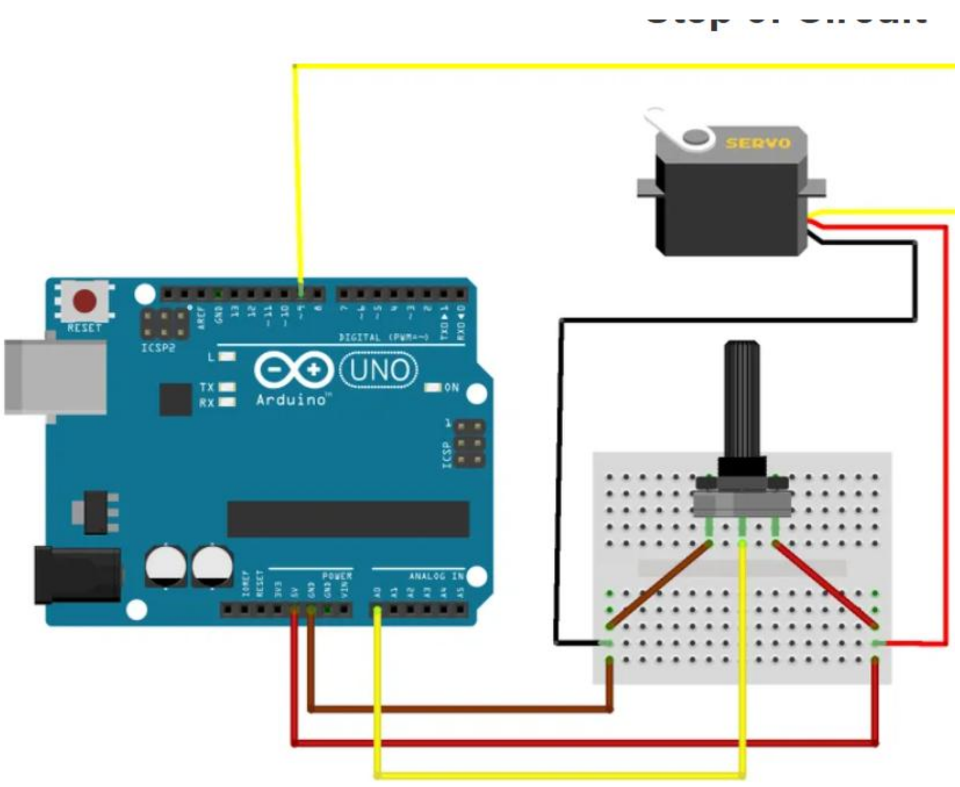


Figure 18 Servo control circuit with potentiometer *(Autodesk, 2025)*

Arduino Code

```
cpp

# include <Servo.h>

myServo Servo;
int potPin = A0; // potentiometer analog pin
int val; // pot value reader variable

void setup () {
myServo.attach ( 9 ); // connect the servo to pin 9
}

void loop () {
val = analogRead (potPin); // read value from 0-1023
val = map (val, 0 , 1023 , 0 , 180 ); // convert to servo angle (0-180)
myServo.write (val); // move the servo
  delay ( 15 ); // small delay for stability
}
```

Simulation Steps

1. Connect all components according to the schematic.
2. Copy the code into the Arduino editor in Tinkercad.
3. Click "Start Simulation".
4. Turn the potentiometer and observe the servo moving at an angle.

Basic Arduino Competency Test Questions

Instructions: Choose the most appropriate answer for each of the following questions.

Chapter 1: Getting to Know the Arduino Board

a. What are the main functions of the Arduino platform?
b. As an operating system for computers.
c. As an open-source platform for creating interactive electronic prototypes.
d. As a programming language specifically for robotics.
e. As a type of battery for electronic devices.

2. The most common microcontroller used on Arduino UNO boards is...
a. ATtiny85
b. ESP32
c. ATmega328P
d. Raspberry Pi Pico

3. Which pin on Arduino UNO is used to read analog signals from sensors?
a. Digital Pins 0-13
b. Pins A0-A5
c. Vin Pin
d. Reset Pin

4. What is the function of the GND pin on the Arduino board?
a. Provides 5V input voltage.
b. As a ground voltage reference (0 Volts).
c. To upload a program.
d. For serial communication.

5. The software used to write, compile, and upload programs to Arduino is called...
a. Arduino Studio
b. Arduino IDE (Integrated Development Environment)
c. Arduino Compiler
d. Arduino Uploader

Chapter 2: Basic Digital Electronics

6. Why do LEDs need to be connected in series with resistors when connected to Arduino output pins?
a. To increase the brightness of the LED.
b. To limit the current flowing so that the LED is not damaged.
c. To change the LED color.
d. To make the LED blink.

7. In a button circuit with an internal pull-up resistor (`INPUT_PULLUP`), what value is read by `digitalRead()` when the button is pressed?
a. HIGH

b. LOW
c. Values between 0 and 1023
d. Uncertain (floating)

8. What is the main function of a transistor (e.g. BJT NPN) when used with Arduino to control a DC motor?
a. As a voltage source for the motor.
b. As a motor speed sensor.
c. As an electronic switch to flow large current to the motor which is controlled by a small signal from the Arduino.
d. As a color regulator for lights on a motorbike.

9. What is meant by the "floating" condition on a digital input pin?
a. The pin is connected directly to 5V.
b. The pin is connected directly to GND.
c. The pins are not connected to a definite voltage level (HIGH or LOW), so the reading is erratic.
d. The pin is sending data.

10. What components are used to build electronic prototypes without soldering?
a. PCB (Printed Circuit Board)
b. Breadboard (Project Board)
c. Multimeter
d. Oscilloscope

Chapter 3: Getting to Know C Language & Basic Algorithms

11. Which function in an Arduino program is only executed once at startup?
a. `loop()`
b. `main()`
c. `setup()`
d. `delay()`

12. The command `digitalWrite(pinLed, HIGH);` is used to...
a. Read the status of the `pinLed` pin.
b. Set `pinLed` as input.
c. Provides HIGH voltage (eg 5V) to `pinLed`.
d. Gives LOW voltage (0V) to `pinLed`.

13. The control structure `if (condition) { ... } else { ... }` belongs to the type of algorithm...
a. Sequential
b. Recurrence
c. Branching
d. Function

14. What is the main difference between `delay()` and using `millis()` for scheduling?
a. `delay()` is more accurate than `millis()`.
b. `millis()` can only be used in `setup()`.
c. `delay()` is blocking (stops the program), while `millis()` allows non-blocking scheduling.

d. `millis()` returns the value in seconds, `delay()` in milliseconds.

15. The `Serial.begin(9600);` function is usually placed inside...
a. `loop()`
b. `setup()`
c. Global variable declaration
d. Homemade function

Chapter 4: Basic Implementation (LED & Button)

16. The code `pinMode(LED_BUILTIN, OUTPUT);` works to...
a. Turns on the built-in LED.
b. Read the built-in LED status.
c. Set the built-in LED pin to send a signal out (control the LED).
d. Set the built-in LED pins to receive incoming signals.

17. If using `pinMode(buttonPin, INPUT_PULLUP);`, the `if` logic to detect a button press is...
a. `if (digitalRead(buttonPin) == HIGH)`
b. `if (digitalRead(buttonPin) == LOW)`
c. `if (analogRead(buttonPin) > 500)`
d. `if (analogRead(buttonPin) < 500)`

18. What is the purpose of the "Blink Without Delay" code that uses `millis()`?
a. Makes the LED blink faster.
b. Make an LED blink without stopping the execution of other tasks by the Arduino.
c. Save Arduino memory usage.
d. Replaces `digitalWrite()` function.

Chapter 5: Reading Sensors & LCD

19. The DHT11 sensor is used to measure...
a. Distance and light
b. Temperature and humidity
c. Air pressure and altitude
d. Current and voltage

20. How to read the value from the LDR sensor connected to pin A0?
a. `digitalRead(A0);`
b. `digitalWrite(A0, HIGH);`
c. `analogRead(A0);`
d. `analogWrite(A0, 128);`

21. The `pulseIn(echoPin, HIGH);` function on the HC-SR04 ultrasonic sensor is used to...
a. Sending trigger pulse.
b. Measures the duration of time the echo pin is in the HIGH state.
c. Calculate direct distance in cm.

d. Set the frequency of ultrasonic waves.

22. What is the minimum number of pins required for Arduino to communicate with a 16x2 I2C LCD module?
a. 6 pins (Data, RS, E, VCC, GND)
b. 4 pins (VCC, GND, SDA, SCL)
c. 2 pins (SDA, SCL)
d. 1 pin (Data)

23. On the Arduino UNO, pins A4 and A5 are specifically used for communication...
a. Serial (TX/RX)
b. SPI (MOSI, MISO, SCK)
c. I2C (SDA, SCL)
d. PWM

24. The `lcd.setCursor(0, 1);` command in the `LiquidCrystal_I2C` library will move the cursor to...
a. Column 1, Row 0
b. Column 0, Row 1
c. Column 1, Row 1
d. Column 0, Row 0

25. What libraries are generally required to use a DHT11 sensor with Arduino?
a. `Wire.h`
b. `Servo.h`
c. `DHT.h` (and possibly `Adafruit_Sensor.h`)
d. `LiquidCrystal_I2C.h`

Closing

Congratulations! You have completed this basic Arduino tutorial. You now have an understanding of what Arduino is, the basic electronic components, how to program it using C/C++, control outputs such as LEDs, read input from buttons, interact with common sensors, and display information on an LCD.

Your Arduino journey has just begun. The world of electronics and microcontrollers is vast and full of exciting possibilities. Feel free to keep experimenting, trying new projects, and seeking additional learning resources from the vast Arduino community. The more you practice, the better you will become at bringing your creative ideas to life.

Keep learning and creating!

Reference

Arduino. (2025, January 1). *Arduino Official Store*. From Arduino Official Store: https://store.arduino.cc/

Autodesk. (2025, January 1). *Instrucable*. From Instrucable: https://www.instructables.com/Arduino-How-to-Control-Servo-Motor-With-Potentiome/

ThinkerCad. (2025, January 1). *ThinkerCad.com*. (AutoDesk) Retrieved January 1, 2025, from https://www.tinkercad.com/

Wokwi. (2019, January 1). *wokwi.com*. Retrieved from wokwi.com: wokwi.com

About the Author

Eko Travada Suprapto Putro, ST, MT is a practitioner and academic in the field of computer engineering and microcontrollers. He completed his undergraduate education in Electrical Engineering at Maranatha Christian University in 1998 and earned a Master of Engineering degree from the Bandung Institute of Technology in 2004 with a concentration in Computer Engineering.

Expertise in C programming, Python, Arduino and Atmel based system development, and MySQL database management. His extensive experience includes various military information system projects and plagiarism detection systems. Among them as an expert consultant for Kodam II/Sriwijaya, Kodam IV/Diponegoro, and Seskoad.

Currently, he is actively writing, teaching, and developing microcontroller-based technical guides for beginners to advanced levels.